Imperial College London

Anomaly detection and generation in time-series for predictive asset maintenance

Hitesh Kumar

Supervised by: Prof. Martin Blunt and Lukas Mosser (Imperial College London) and Tolu Ogunseye (Shell Research Limited*)

Project report submitted for the MSc in Applied Computational Science and Engineering (Module: ACSE-9)

Submitted on 30/08/2019

Email address: hitesh.kumar15@imperial.ac.uk

College Identification Number: 01058403

GitHub alias: FistOfHit

*Shell Research Limited, Shell Centre, London, United Kingdom, SE1 7JN

Anomaly detection and generation in time-series for predictive asset maintenance

Hitesh Kumar[*]
Department of Earth Science and Engineering
Imperial College London

Abstract

Identifying failures and damage (anomalous behaviour) in remote mechanical assets such as compressors is of great importance to many industries. As the number and complexity of such machinery increases, automated methods that can make inferences from the multivariate time-series produced by sensor readings from these assets are required. Here we show that LSTM (Long Short-Term Memory) auto-encoders are able to prepare intervals of these multivariate time-series for classification into anomalous and normal behaviour, and are able to use the easily-interpretable nature of this model to identify sets of features which contribute to anomalous behaviour. Additionally, we demonstrate that LSGANs (Least-Squares Generative Adversarial Networks) are able to learn the static (non-time dependent) marginal distributions of the time-series and samples can be drawn from this to generate synthetic data.

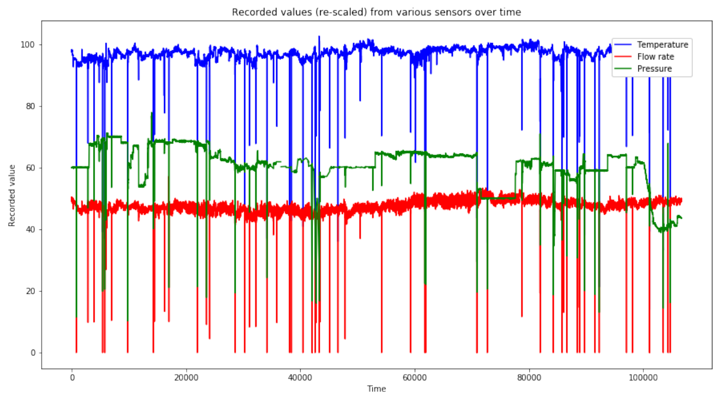

Fig. 1: A multivariate time-series formed from sensor recordings (features) from a LPC. Each feature has been re-scaled and unique identifiers that could indicate where in the LPC and when these readings were taken have been removed for security and anonymity.

1 Introduction

Predictive asset maintenance (PAM) is of great importance to all industries using large-scale machinery, and in particular, for crude oil extraction. On oil extraction platforms, Low pressure compressors (LPCs) are tasked with increasing the pressure of gases for more efficient transportation from the extraction source. Monitoring the state and input/outputs of the compressors in real time is performed via sensors that use various methods to measure and report the values of certain variables at given intervals of time, and hence producing multivariate time-series data. We see an example of this in figure 1 where three re-scaled and anonymised variables for a LPC over a period of time are displayed. We will refer to each of these as a feature, and its clear that each one exhibits different seasonality and trends.

However in rare instances LPCs have been observed showing various forms of anomalous behaviour, and even more rarely this behaviour precedes partial or total shutdowns on the platform. One such issue with compressors include stonewalling or ”choking”, occurring when the flow rate is very high yet the discharge pressures are low as the resistance to the flow in the output drops. Another cause for dangerous anomalous behaviour is when the flow rate becomes very low and a process known as surging occurs, resulting in a breakdown of steady flow within the compression chamber and a possibly damaging increase in temperature, vibration amplitudes and even flow reversal. These processes can cause a significant reduction in useful compressor output, a decrease in efficiency or even result in parts that are damaged to the point of requiring replacement.

Large financial losses due to damage and downtime provide ample motivation for predicting such instances of very rare anomalous behaviour. Avoiding such losses requires either sensors able to automatically take actions based on pre-defined thresholds, or supervision by experienced compressor engineers. Automating these inferences is a

[*]Project in collaboration with Shell Research UK

non-trivial task, and requires models that can learn the complex patterns and inter-dependencies in variables such as temperature, pressure and flow rate etc. being recorded by multitudes of sensors simultaneously.

Anomaly detection algorithms have been applied to many fields such as finance [Ahmed et al., 2016], medicine [Schlegl et al., 2017] and engineering [Bates et al., 2017] to name a few. Traditionally, modern statistical methods such as SVMs [Cortes and Vapnik, 1995] (support vector machines), isolation forests [Liu et al., 2008] and clustering algorithms [Münz et al., 2007] etc. have been applied to the task and have demonstrated capability in handling high dimensional data whilst requiring relatively few observations to be fitted reliably when compared to deep learning methods [Caruana and Niculescu-Mizil, 2006]. However these approaches are lacking in two areas; using temporal dependencies in time-series data, and having easily interpretable of their results. In many PAM applications, the data requiring classification is time-dependent, and whilst there are methods to take time dependency into account, they require the user to define the nature of the dependency or the intervals in time that the models can use to make inferences [Rüping, 2001]. Interpretability is an important issue, as models that make use of complex non-linear kernels such as SVMs apply these non-linear transformations to data [Lin and Lin, 2003] that makes determining the cause of anomalous classifications very difficult.

Deep learning approaches to anomaly detection have also been explored, and many experiments have been performed in detecting and generating anomalies in time series, using various approaches such as recurrent variational auto-encoders (VAEs) [Park et al., 2018] and temporal convolutional networks (TCNs) [Zhou et al., 2016]. Projects such as [Gugulothu et al., 2017] demonstrate that latent space representations of multivariate time series recorded from pumps, generated using auto-encoders can be used to make inferences on the "health" of the machines, and whilst the model is made robust to missing values, the data-set used seems to be of far higher quality than what is available to us, which may be what allows them to make inferences from the latent space vectors.

For generating data, [Mogren, 2016] and [Esteban et al., 2017] show that recurrent generative adversarial netowrks (GANs) [Goodfellow et al., 2014] can indeed learn the static distributions of musical sequences and medical data, but offer little or no discussion on the temporal dependencies in these random samples drawn from learned distributions. [Graves, 2013] discusses a less statistical and more direct approach to time series generation in the form of handwriting synthesis, again using recurrent neural networks, but to make direct inferences on future data using samples to initialise the generation. The publications above discussing the use of recurrent layers in GANs all seem to have little theoretical justification as to why simply adding recurrent layers to either of the discriminator/generator would result in the generation of time-series with the same or similar temporal dependencies as the real data, as opposed to random samples.

It is clear that a rich variety of literature is available on the topics being explored in this project. Results and ideas from various works are used throughout this project, and are referenced where mentioned. We explain the exact concepts and methods used in more detail in our methodologies in sections 3 and 4, but summarise them below.

1.1 Approach summary

In this project, we implement an auto-encoder containing LSTM recurrent layers, in order to learn a spatially compressed (no compression in time), latent space representation of a multivariate m-dimensional time series of length N, $\{\mathbf{X_t}\}_{t=1}^{N}$, $\mathbf{X}_t \in \mathbb{R}^m$ whilst capturing temporal dependencies, and then using a time series in the latent space to reconstruct the inputs. This is discussed in further detail in section 3. Additionally, we implement methods that allow for identifying the causes of each anomalous classification, at the level of individual sensors, allowing for a much deeper understanding of the nature of the anomalies and for the potential of further fine-grain classifications.

We also attempt to overcome the other issue with many applications of anomaly detection, the lack of training data, by using generative adversarial networks to learn the static, non-time dependent probability mass functions $p(x)$ for $x \in \mathbb{R}$ of recordings from individual sensors. We also discuss possible methods to post-process samples from the generators into realistic time-series that resemble recordings from sensors, rather than just being random samples drawn from static distributions. We discuss both of these points further in section 4.

1.2 Requirements and SDLC

The repository for all code finalised during this project is available at: github.com/msc-acse/acse-9-independent-research-project-FistOfHit, with the project being under an MIT License, but access to the repository is controlled by the msc-acse group administrators. The code submitted to this repository does not contain the dataset used in this project and does not provide the unique identification numbers of the sensors corresponding to the LPC. The project was not version controlled and planned in this GitHub repository but instead in an internal Azure Devops repository for security and so access to commit history and older versions of the data is restricted.

The entirety of the project was developed in Python 3.7.3, using standard packages included in Pylab, as well as other open-source packages, most notably PyTorch, Pandas and SkLearn, with the CUDAToolkit 10.0 and CUDNN libraries to utilise local GPUs for accelerated deep learning. More details are included in the project README in the repository. The choice of development and experimentation environment was the Jupyter Notebook, highly configurable and well suited to quick testing and easing incremental production. However for finalising operational code and scripting the solutions, the Spyder3 IDE was used, with more advanced code style checks and profiling functionality.

The development methodology practised during this project was a mixture of both waterfall and agile. We combine the two approaches and focus on documentation and iterative improvements after prioritising the development of minimal working solutions first, but restricting this to sub-sections (Exploratory data analysis, Data pre-processing, Training etc.) of the project, and progressing through these sub-sections in a linear fashion. This allowed for a simple and intuitive progression through the project but also to allow short-term flexibility for extensive and quick experimentation at a unit level.

2 Data exploration and pre-processing

The data-set provided consisted of readings from 405 distinct sensors, reporting variables such as temperature, pressure, flow rate and many others for various components of a large crude oil extraction pipeline, of which the LPC was a component. Of these 405 sensors, there were 227 that were specific to the LPC and a list of these was provided. Knowing that these readings all come from one asset however implies the scope for significant dimensional reduction, as many features are likely to be highly correlated and hence contain very similar information. However rather than dimensionality reduction via embedding or clustering algorithms, it was clear that the number of features could be reduced significantly simply by filtering out poor features. This is discussed further in section 2.2. Each data point is time stamped with each stamp separated by 10 minutes, but not labeled, however nine specific dates for anomalies that caused shutdowns in the LPC were provided and from this, labels could be produced to suit the problem and model. What was available contained observations spanning 916 days (∼2 years and 6 months) or approximately 130,000 individual readings per sensor.

When briefed on the data-set by engineers, we find that certain periods of time correspond to offline recordings, where the compressor was shut down and hence certain variables such as temperature and pressure will certainly be below a threshold, requiring a filtering in time also. We expand on this in section 2.1.

2.1 Filtering in time

Thresholds imposed on the readings from certain sensors such as temperature and pressure were provided by the engineering team operating the LPC, and whenever any of these sensors exceeds the given thresholds, that data (as well as small windows of time around them) would be deemed ”offline” and would be removed from all features, leaving gaps in time for the entire data set. This issue was prevented by using the assumption that the compressor returns to the same ”normal” behaviour after a short period of time after being activated that it could have been considered operating at before any anomalous behaviour was detected. This assumption was valid according to the engineers that operate and maintain the LPC, and so the data was simply appended whilst taking care to preserve the original timestamps. This resulted in a $\sim 24\%$ reduction in the number of data points available.

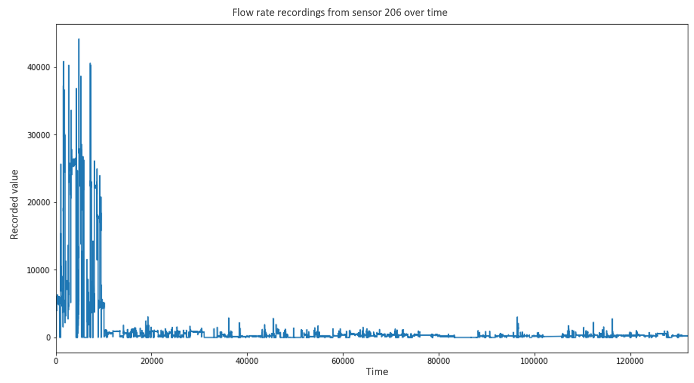

Fig. 2: Feature No.206 (Flow rate), showing where the scale of the data changes dramatically and remains as such for the rest of available time

Another issue we find is that for some sensors, the data recorded seems to show sudden changes in scale, for large periods of time, such as in figure 2. Upon investigation, this was caused by LPC engineers monitoring the data and deciding to change the units that the recordings were taken in, and since no information was available to invert these conversions, it was decided to use the differences between consecutive points in time ($\widehat{\mathbf{T}}_t = \mathbf{T}_t - \mathbf{T}_{t-1}$ where $\mathbf{T}_t, \widehat{\mathbf{T}}_t \in \mathbb{R}^m$) rather than the values themselves as our features. This would, to a large extent, prevent this issue with changes in scale and also would apply a linear de-trending to our data, removing some of the burden on our model for being able to capture very long term dependencies in time.

2.2 Filtering in space

When observed, we see that some sensors clearly have disproportionately large periods of consecutive missing values (reported as NaN) and some seem to have a very large number missing values many over the entire time period despite

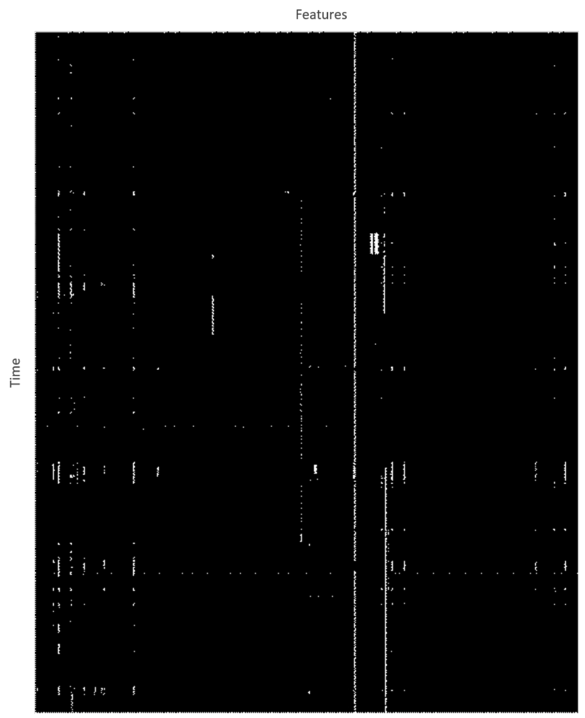

Fig. 3: Heat map of available data with features on the horizontal and time on the vertical. White indicates missing values (NaN recordings).

them rarely occurring at consecutive time-steps (see figure 3). Such behaviour would make interpolation less reliable and hence adversely affect the inferences made from the data. Also, when observing basic statistics of these features, we see that some sensors report 0 rather NaN repeatedly, for large periods of time, and this is not detected by previous simpler missing value detection. Other issues included general lack of information (low variance regardless of scale) in some features relative to others and the issue of some features being semi-categorical in nature, reporting a relatively few number of unique values, such as sensors on valves or pumps. After some experimentation, it was decided that four criteria should be applied and we would remove any feature that had:

- More than 1% of its values missing
- More than 12 consecutive missing values
- Less than 2000 unique values
- Variance lower than 1

Once these were applied, we were down to 111 features. With no further suggestions on features to remove or keep from the LPC engineers, we saw no need for any further feature reduction. However for the second part of this project, where the probability distributions of the data are very important, we had to enforce a fifth criterion:

- More than 144 consecutive repeated values

This served mainly to prevent extremely skewed distributions when training generative models, and 144 was chosen as the threshold as it is exactly one day in time-steps. Hence if a certain sensor was not reset or was faulty even after a daily assessment by engineers, we would not include it in our data set. After this filter was applied, our second data set for generative models had 62 features. We discuss similar issues with the data further in sections 2.3 and 4.1.

We now use two separate data sets produced from the original super set: the ”detection” data set, with 111 features and the ”generation” data set with 62 features, both having 106,708 individual time-steps. These files were stored in CSV format, with each individual reading being stored in memory (once loaded using pandas) with the numpy float64 data type.

2.3 Other pre-processing

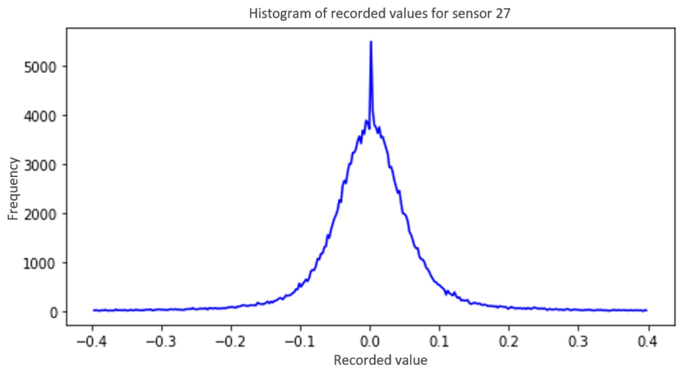

Fig. 4: The static distribution of the central 98th percentile of the normalised values of feature No.27 (Input pressure) over all time. We observe a seemingly normally distributed variable with a strange peak near 0, due to the large number of repeated 0 recordings, and heavy tails on either side.

Other transforms were experimented with, however the natural distributions of the data had to be taken into account when using any normalisation procedures. The vast majority of features were not distributed normally, with the central 98th percentiles following a somewhat normal distribution, whilst the data at the two extremes lying on light, but large tails (figures 4 and 5).

Investigating further, we find that our distribution is leptokurtic, displaying an excessively large positive kurtosis or ”tailedness” (fourth moment) when compared to normal distributions. We see a direct comparison of the two in figure 6 where we see the visual difference between the two distributions as well as their respective sample kurtosis estimates, and we see that the value for our data’s distribution is significantly higher than it should be were our data set distributed normally. Such distributions are referred to as ”super-Gaussian”, and more appropriate models to be fitted to our data would be either students-t or laplace distributions. We do not investigate this further here, and do not require to

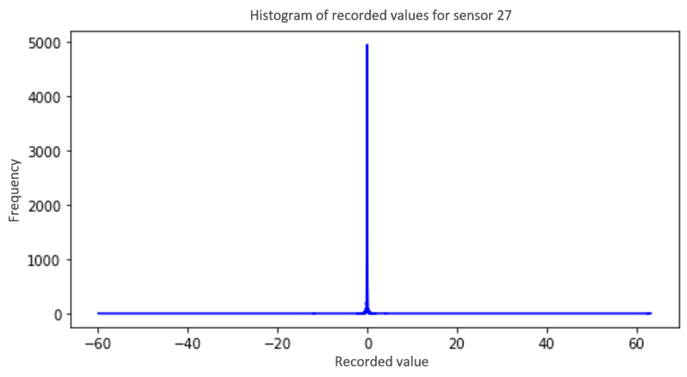

Fig. 5: The distribution of all of feature No.27 (Input pressure) over all time. This bares almost no visual similarity to the previous distribution due to the one percentiles on either end being relatively large values. We see later that metrics used to assess distribution similarity also struggle to see that these two distributions are almost exactly the same.

in section 4.4 either as a better understanding of this may have little effect on the outcomes of the project. These asymmetric

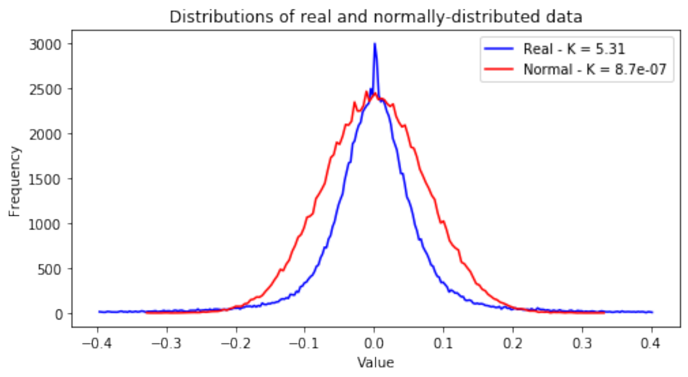

Fig. 6: Histograms of both the data set available to us and a normal distribution with the equivalent mean and variance. We see clearly that our distribution has heavier symmetric tails and larger probability mass near the mean. The sample kurtosis estimates for both distributions are given in the labels under the value of "K", with our data exhibiting a value of 5.31 as opposed to the near zero result from the normal distribution.

extreme value distributions would not behave well after simple $[0,1]$ min-maxing, and since the variances measured (on the assumption that the distributions were normal) varied from a range of 1 up to 10^4, it was decided to rely on the differencing in time to remove many of the scale changes causing this disparity in variances, and then standardising the features by centering and scaling by estimated statistics:

$$\mathbf{X}_i = \frac{\widehat{\mathbf{T}}_i - \mu}{\sigma} \tag{1}$$

Where

$$\mu = \frac{1}{N}\sum_{i=1}^{N} \widehat{\mathbf{T}}_i \tag{2}$$

$$\sigma = \sqrt{\frac{1}{n}\sum_{i=1}^{n} (\widehat{\mathbf{T}}_i - \mu)^2} \tag{3}$$

After all of these operations, we divide the data-set by grouping the time-series into 144 (one day) time-step long sequences, strided by 72 time-steps. Deciding this sequence length and stride proportion required various tests, and it was determined that this sequence length was appropriate to keep computational costs reasonable, as well as being long enough to allow the LSTM layers to learn and recreate the short-term temporal dependencies.

These sequences were then placed into mini batches of size 32 for training/testing purposes and batch sizes of 1 for assessment and presenting. PyTorch's DataLoader implementations were used for creating these batches and for creating iterators to make it easier to pass data through the models. Training, validation and test splits were created in the ratio 70:15:15 and for final testing purposes the training and validation sets were merged for training anomaly detection models, however for anomaly generation models, we did not create such splits.

The splitting of the data set into train, validation and testing was done in time, not randomly, so that the first 70% of data occurring in time would be training, the next 15% would be validation and the final 15% as test data. This splitting scheme was necessary with strided sequences to prevent overlap between the sets, which would defeat the point of splitting the data in the first place. A potential issue with this scheme is that we would have to assume the data did not have any significant long term trends (spanning the dataset) otherwise our models would have to take this into account too. We discuss this issue further in section 3.4.

3 Anomaly detection

3.1 Methodology and background

There were many options when deciding what approach would suit the data best, but there were two clear routes: supervised or unsupervised. On supervised learning, experiments were performed on using LSTM-based models for direct inference, both classifying individual time-steps and intervals directly. This however suffered from the main issue with all anomaly detection problems, which was the lack of labeled anomalous data [Haixiang et al., 2017]. With 9 labeled points in time, even by using reasonably wide windows or multiple overlapping windows (see figure 7), the class imbalance would still be beyond simple up-sampling techniques. To mitigate this, we attempted to see if remaining useful life (RUL, or alternatively, "Time to failure" - TTF) could be predicted, using continuous labels that span the entire data-set, converting this from a classification to a regression problem. This approach yielded no results either as all models failed to realise any indicators - if there were any - that would allow such a prediction.

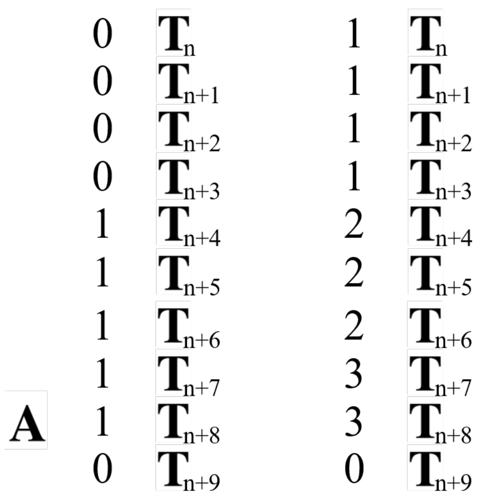

Fig. 7: Single window (left) and multiple window (right) labelling schemes for the data-set. The labelled anomaly is at time-step A. Labels are the numerical values assigned to each time-step, $\mathbf{T}_i$ Various window widths and overlaps were experimented with.

This warranted an unsupervised approach, where class imbalance would no longer be an issue and one possibility was to use an auto-encoder model. Auto-encoder models [Hinton and Zemel, 1994] are usually feed-forward multi layer perceptron (MLP) models with the purpose of first encoding the input $\mathbf{X} \in \mathbb{R}^m$ with an encoder model in an unsupervised manner into some latent space representation, $F_E(\mathbf{X}) \in \mathbb{R}^l$. These latent vectors are then decoded back into input space with a decoder model $F_D : \mathbb{R}^l \to \mathbb{R}^m$, in such a way as to reconstruct the input:

$$\hat{\mathbf{X}} = F_D(F_E(\mathbf{X})), \quad \hat{\mathbf{X}} \in \mathbb{R}^m \tag{4}$$

The reason for using such a model is that applying a constraint where $m > l$ forces the model to learn to compress the data in a meaningful way, so as to retain enough information in the latent space vectors to recreate the data as accurately as possible according to whichever similarity metric is used. Models where $l \geq m$ also have their uses and advantages [Bengio et al., 2009], but these are not discussed in this project.

LSTMs [Hochreiter and Schmidhuber, 1997] are a form of recurrent neural network where information from previous inputs is stored in memory, in an internal state, and is able to influence the outputs of the model at each time-step thereafter. This essentially allows the model to retain, forget and learn information through time as more inputs are presented to it, and hence naturally lends itself to learning temporal dependencies in time series. Mathematically, an LSTM layer uses a series of gates in the order:

$$\mathbf{F}_t = S(W_f\mathbf{X}_t + U_f\mathbf{H}_{t-1} + b_f) \tag{5}$$
$$\mathbf{I}_t = S(W_i\mathbf{X}_t + U_i\mathbf{H}_{t-1} + b_i) \tag{6}$$
$$\mathbf{O}_t = S(W_o\mathbf{X}_t + U_o\mathbf{H}_{t-1} + b_o) \tag{7}$$
$$\mathbf{C}_t = \mathbf{F}_t \circ \mathbf{C}_{t-1} + \mathbf{I}_t \circ T(W_c\mathbf{X}_t + U_c\mathbf{H}_{t-1} + b_c) \tag{8}$$
$$\mathbf{H}_t = \mathbf{O}_t \circ T(\mathbf{C}_t) \tag{9}$$

Where $\mathbf{X} \in \mathbb{R}^m$ and $\mathbf{F}_t, \mathbf{I}_t, \mathbf{O}_t, \mathbf{C}_t$ and $\mathbf{H}_t \in \mathbb{R}^h$ lie in the "hidden" or output space (this implies that LSTMs themselves are also capable of compressing data in space to an extent). $\circ$ represents element-wise multiplication and T the element-wise tanh function. The element-wise sigmoid function S defined as:

$$S(\mathbf{X}) = \frac{1}{1+e^{-\mathbf{X}}}, \quad S(\mathbf{X}) \in \mathbb{R}^m \tag{10}$$

is used to assign a $(0,1)$ weight to first decide which components of a sum of the input at time t, $\mathbf{X}_t$ and previous output $\mathbf{H}_{t-1}$ (and a bias vector) to forget, as well as which components to retain in memory ($\mathbf{I}_t$) and which elements to output next ($\mathbf{O}_t$). These gates are then used to calculate which elements of the previous cell state or memory, $\mathbf{C}_t$ to forget and to learn for the next step, according to (8). The output at time t is then calculated using the output filter and the current cell state with (9). For a more rigorous and detailed explanation of the mathematics and intuition here, see [Hochreiter and Schmidhuber, 1997].

The final decision was therefore to train an Auto-encoder containing LSTM layers to compress a high-dimensional time-series of length N, ($N = 144$ was finalised after experimentation) $\{\mathbf{X}_i\}_{i=1}^N$, $\mathbf{X}_i \in \mathbb{R}^m$ into a low-dimensional representation $\{\mathbf{x}_i\}_{i=1}^N$, $\mathbf{x}_i \in \mathbb{R}^l$ where $l < m$ whilst implicitly learning time dependencies between time-steps in the latent space in an unsupervised manner, and then decompress these representations to generate reconstructions, $\{\widehat{\mathbf{X}}_i\}_{i=1}^N$. Once generated, the log-mean reconstruction error could then be calculated:

$$\varepsilon_i = \log\left(\frac{1}{N}\sum_{i=1}^{N}||\mathbf{X}_i - \widehat{\mathbf{X}}_i||_2^2\right), \quad \varepsilon_i \in \mathbb{R} \tag{11}$$

The logarithm was taken purely for visual effect when observing the reconstruction errors themselves, and did not affect the results in any way. It remained now to distinguish the set of reconstruction errors into anomalous and normal. The initial idea was to use a two-step process, the first being using the lowest reconstruction error of all labeled anomalies to set the initial threshold, resulting in all sequences reporting lower reconstruction errors labeled as ”safe”.

The second step was then to use 1.96 standard deviations (to exclude 5% of this subset) from the mean of these ”safe” errors as the new threshold for defining a new and more robust boundary on the reconstruction errors which could be used to make the final classification between normal and anomalous. However, upon observing the distribution of the reconstruction errors (see figure 8) the assumption was made that this was either a composition of two distinct distributions or a heavy-tailed extreme value distribution. Based on this, it seemed a more effective and simpler method to determine a reasonable threshold was to use a certain percentile of the distribution. This had the additional advantage of allowing for expert judgment on the trade-off between how extreme an anomaly should be versus how often a warning would be raised. Ultimately, the 85^{th} percentile was chosen as the threshold.

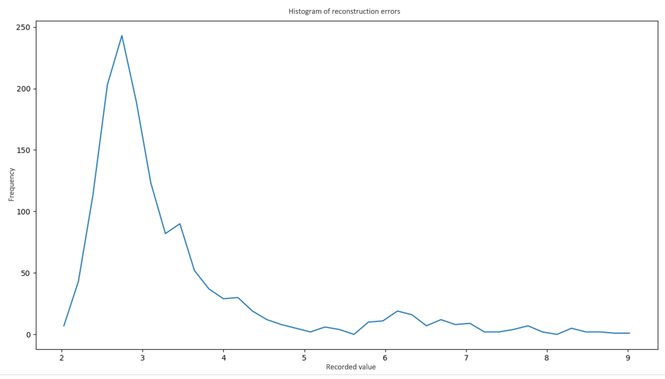

Fig. 8: Histogram of all of the log reconstruction errors, here we see what seems to be a log-normal distribution with a heavy tail, but could be a variety of other distributions with certain parameters, as many share this shape.

In operational use, sequences of a fixed length could be created every time new information is available from the sensors and passed through the model to determine the reconstruction error for the interval of time spanning from the present to 24 hours before. We did not develop such infrastructure in this project however.

3.2 Model architecture

The overarching idea here was to implement unsupervised ”lossy” compression and decompression in feature space, and so the auto-encoder containing two LSTM units was developed with the latent space having a lower number of dimensions than the input space. As depicted in figure 9, the model takes a N-length time-series of m-dimensional vectors $\{\mathbf{X}_i\}_{i=1}^{N}$. This is passed through two fully connected linear ”compression” layers, each reducing the dimension of the vectors by the same ratio so that the reduction in size from the input space to latent space was linear through all encoder layers, and vice-versa for decoder layers. The non-linearity applied to these two layers was LeakyRelu [Xu et al., 2015] with the negative gradient being 0.01. The final component of the encoder is a LSTM layer which performs the final dimensionality reduction to generate a time-series of N, l dimensional vectors. The element-wise tanh non-linearity is applied to each latent vector in order to enforce soft constraints on the size of the values in these vectors, and allowing for a form of normalisation to ease training for the decoder's LSTM-layer.

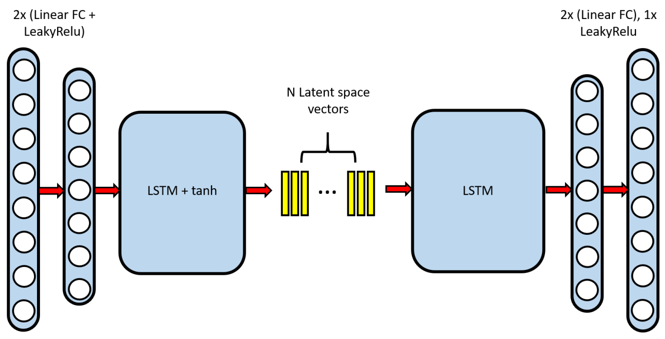

Fig. 9: LSTM-Autoencoder architecture, here we choose to keep all N outputs in time from the encoder rather than just the last as our latent vectors

The decoder consists first of a LSTM-layer which increases the dimension of the latent vectors by a fixed ratio (As described before), and then two fully connected linear layers following the same ratio but now expanding the number of dimensions of the vectors back to the size of the feature space. The non-linearity is applied to the first of these linear layers. For the number of input features specific to this task, a latent space size of 60 was found to be the near-optimal size for the best reconstruction errors whilst providing adequate compression (from 111 input features) for the model to learn robust features in the time-series. For training, after optimising hyper-parameters via grid searches, the optimiser used was RMSprop [Tieleman and Hinton, 2012] with the ”Centered” [Graves, 2013] option enabled and a learning rate of 10^{-5}.

Table 1: Confusion matrix generated from decision boundary, assuming only labeled anomalies are negatives. Whilst the number of false positives is relatively high, there is only one false negative, and that too is very close to the boundary. A low false negative rate is desirable in predictive asset maintenance, where flagging any abnormality is preferable to allowing one to propagate and possibly cause a shutdown.

Actual \ Predicted	True	False	All
True	1203	205	1408
False	1	8	9
All	1204	213	1417

3.3 Results

The main purpose of the LSTM auto-encoder was to reconstruct time-series, and to this end, we see that it convincingly does so for many sequences. For those with the lowest reconstruction error we see a clear understanding of the patterns and statistics of the sequence from the model, an example being figure 10. From the distribution of the reconstruction errors (figure 11) of this particular sequence, we observe a slight positive bias but otherwise strong evidence that they are normally distributed indicating that the errors are just noise as opposed to a systematic issue).

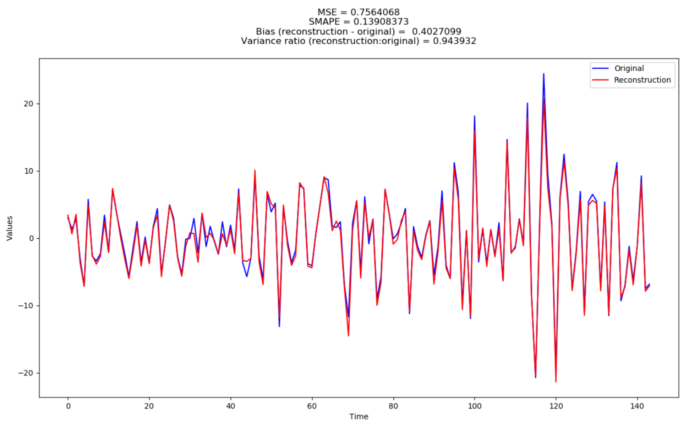

Fig. 10: Plot of a real (blue) and reconstructed (red) time-series of length 144 from feature 5 (something). We see that both the mean square error (MSE) and Symmetric mean absolute percentage error are quite low relative to the scale of the data, indicating a good reconstruction. Additionally, the variance ratio of the two time series are very similar, proving that the model has learnt the time-dependent statistics (scale and shape etc.) of the data. We do observe a slight bias however.

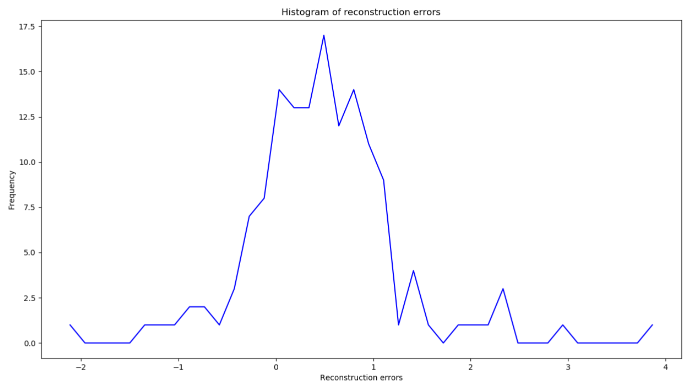

Fig. 11: Histogram of the reconstruction errors from the data in figure 10. We see a reasonably normal distribution, but not centered at 0 implying a slight bias. The average bias of this particular feature however is 0.24.

However, given the variance in reconstruction losses as shown in figure 12, the model does indeed struggle with some sequences. The main task however was to classify these reconstruction errors into normal and anomalous. Table 1 shows the confusion matrix generated from the final classification after reconstructing the sequences and then classifying. A net accuracy of 85.5% indicates that the model is indeed able to correctly identify the majority of normal behaviour. More importantly, we see indeed a distinct separation between anomalous and normal reconstruction errors in figure 12, and our model agrees with this as we see in figure 13, qualitatively indicating a sensible decision boundary. When plotting the Receiver operating characteristic (ROC) curve (figure 14), we see that the model is indeed very capable of having a very high specificity without compromising an unreasonable amount of sensitivity, evidenced by the area under curve (AUC) of 0.92.

Due to only having nine labeled anomalous sequences, we should be careful about conclusions made from the confusion matrix and ROC curve, and ideally for a more robust decision on the ability of the model, a larger sample size of anomalies should be used. However, observing the reconstruction errors and the results of the classification suggests that there may be many more anomalies present in the data set that were not labeled or detected initially by LPC engineers as they did not cause automated shutdowns or did not damage the LPC enough individually to be noticeable.

Another interesting point to note is the clustering in time of the labeled anomalies. This may have an effect on the reconstruction error as we observe that anomalies 3 to 6 occur nearby in time, and may have some temporal relationships despite the fact that the LPC was shut down and restarted from what we have been told to assume is a normal state. In other words, the assumption that these anomalies are independent from each other time may not be true, and such a relationship may affect the reconstruction errors and hence classification results. This is not included in section 3.4 under further work however, as here we ensure that there is no overlap between sequences that precede anomalies and those that contain anomalies, and so investigating this would fall under investigating the compressor itself.

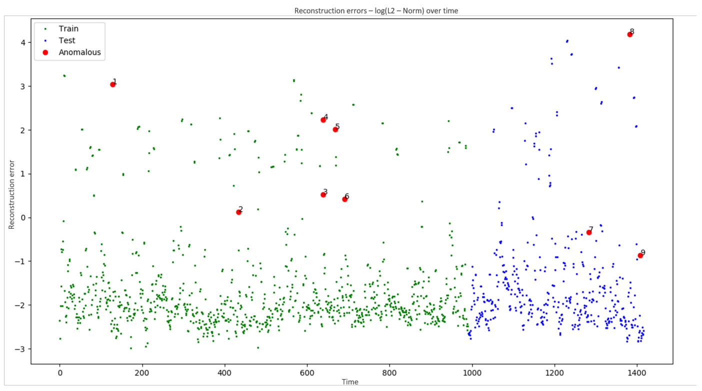

Fig. 12: Reconstruction errors of all 1414 sequences, split into training and test sets. The red numbered points are the labeled anomalies, numbered in the order in which they occur. We see that all anomalies show some pattern of having high reconstruction errors as compared to the majority of the data.

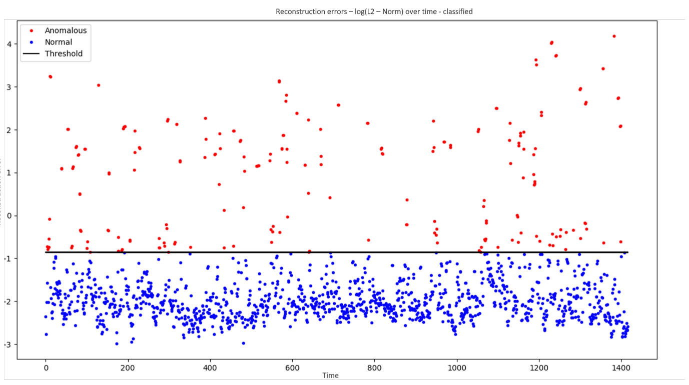

Fig. 13: Final classification of all reconstruction errors, confusion matrix for this in Table 1. We see that the classification by the model includes 8 out of 9 labeled anomalies in its set of anomalous sequences, indicating a sensible result.

Additionally, we experimented with clustering in various 2-dimensional projections and embeddings of the latent vectors encoded from the input vectors, using methods such as linear and non-linear kernel principal component analysis (PCA) and t-distributed stochastic neighbour embedding (t-SNE) (see figure 15) but no experiments ever provided any evidence for patterns or clusters naturally emerging from this and so this line of experimentation was abandoned in

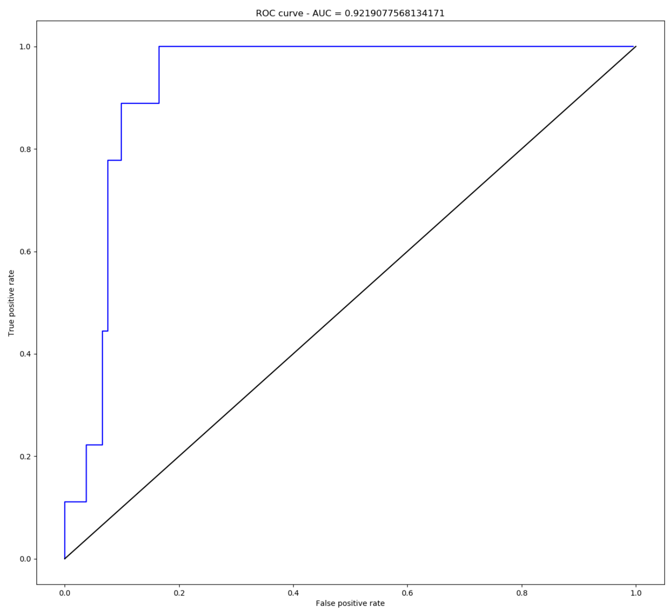

Fig. 14: ROC curve for the model, the large AUC indicates that the model is able to reach 100% specificity without sacrificing too much sensitivity.

favour of evaluating the reconstruction errors instead, but other sources do indicate that meaningful embeddings can be found from certain data-sets/models [Gugulothu et al., 2017].

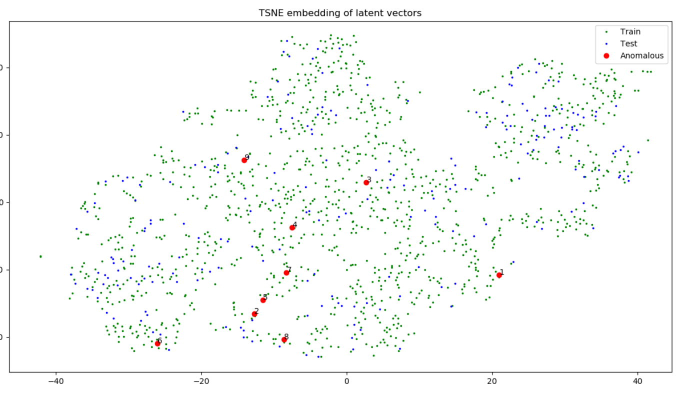

Fig. 15: t-SNE embedding of all reconstruction errors, red points are labelled anomalous sequences, numbered in the order in which they occur in time.

3.4 Conclusions

From the results found, we can see that LSTM auto-encoder models are indeed able to reconstruct intervals of multivariate time series data. We show that the mean reconstruction errors do exhibit patterns, and by setting an appropriate threshold based on the distribution of the errors, the majority of intervals that are labeled as anomalous are indeed classified as anomalous whilst maintaining a very low false negative rate. We have discussed some possible issues with this conclusion, and we now present possible solutions for these in the next sub-section.

3.5 Further work

There are 3 main areas in which further work can be done:

- Adding a time-dependency to the threshold determination function, for a decision boundary based on local reconstruction errors in time, as opposed to the current definition, defined globally for all time. This could be as simple as using a subset in time of the n reconstruction errors

$$\{\varepsilon_i\}_{i=j}^{t} \quad \text{where} \quad t \leq n, j \geq 1 \tag{12}$$

  and determining a local distribution $P_j^t(\varepsilon)$ of the errors to base the threshold on, or as complicated as fitting a Gaussian process regression model to determine continuous mean and variance functions for the data.

- Predicting the remaining useful life (RUL, or alternatively called time to failure, TTF) of the asset using some form of regression on the time series made by the reconstruction errors to predict when the errors will exceed the defined threshold(s). Further investigation is pending into what behavior exactly the reconstruction errors display over time as the compressor runs. The approach could involve fitting statistical models or other LSTM-based neural network models to the reconstruction error time series to make RUL predictions. This would naturally require addressing the earlier point of having infrastructure to process data in real time.

- Determining the causes of individual instances of anomalous classification. Possible approaches could be a combination of recording reconstruction errors for each feature instead of the mean to more precisely isolate the subsets of features responsible for a certain reconstruction error. This may require significant refactoring of parts of the code base however.

4 Anomaly Generation

4.1 Methodology and background

The approach to generating anomalous time-series data consisted of three distinct steps. The first was to train a generative model to learn and produce random samples from an approximation of the probability distribution of the data. The second was to then ensure that the N random samples drawn from this generative model could be considered a realistic time-series of length N as determined by either some metrics or by expert engineers. The final step would be to then generate such time-series so that anomaly detection models would classify them as anomalous.

For the first step, we initially considered purely statistical methods such as finding an approximation for the inverse normalised cumulative distribution function, $F^{-1}(x)$ for $x \in [0,1]$, and then sampling from a uniform distribution to pass those samples through this approximated function. But there were no guarantees on this for how well a given interpolation or curve fitting method would work given that we had arbitrary, possibly discontinuous and very extreme-valued distributions to work with, and adding the additional constraint being that this approximation had to be invertible meant that a very limited number of methods were left.

After experimentation, it was decided that training a generative adversarial network (GAN) to learn a direct mapping between a multi-dimensional unit gaussian $\mu \in \mathbb{R}^l, \ \mu_i \sim N(0,1)$ (the latent space) and the probability mass function $p(\mathbf{X})$ where ($\mathbf{X} \in \mathbb{R}^m$ and $m \geq l$ usually) of the real distribution, estimated by the sample available to us (The choice of using a gaussian distribution for sampling from the latent space as opposed to a uniform was recommended by [White, 2016]).

GANs work by having two networks, a discriminator D which is tasked with discriminating samples as coming from the distribution of the real data, $p(\mathbf{X})$ or from a synthetic generated distribution, $p_G(\mathbf{z})$ generated by a generator G. Various types of loss functions are discussed shortly that allow the discriminator to learn how to distinguish one from another, but they all revolve around maximising the probability that it correctly identifies a given sample as coming from a particular distribution. The task of G is then to map random samples from a latent space distribution $p(\mathbf{z}), \ \mathbf{z} \in \mathbb{R}^l$ to an output which can be considered as being sampled from $p_G(\mathbf{z})$.

Based on the same loss function, G then is tasked with optimising its mapping (and hence its output distribution) to minimise the distribution similarity loss between $p(\mathbf{X})$ and $p_G(\mathbf{z})$. Optimising D and G occurs in iterations with both taking turns to achieve their own objectives, but ideally this process should ultimately result in D being unable to distinguish the real and generated distributions as G learns to output a distribution that minimises the similarity loss being measured. A more detailed description of this process is provided in [Goodfellow et al., 2014].

Of the various choices of loss functions to train a generative adversarial network with, there were four main variants implemented and tested in this project. The first is the original GAN model which has its loss based on minimising the Kullback-Leibler (KL) divergence. We use this loss, but with a modified loss function for the generator

G [Goodfellow et al., 2014] to provide better gradients to train the discriminator, D:

$$\mathcal{L}_D = -\frac{1}{N}\sum_{i=1}^{N} + \log\left(S(D(\mathbf{X}))\right) + \log\left(1 - S(D(G(\mathbf{z})))\right)$$
$$\mathcal{L}_G = -\frac{1}{N}\sum_{i=1}^{N} \log\left(S(D(G(z)))\right)$$
$$\mathbf{X} \in \mathbb{R}^m, \quad \mathbf{z} \in \mathbb{R}^l \tag{13}$$

Where m and l are the sizes of the feature space and latent spaces respectively, and S is the element-wise sigmoid function described in (10). As well as this, we also experimented with using the least-sqaures loss function:

$$\mathcal{L}_D = \frac{1}{N}\sum_{i=1}^{N}(D(\mathbf{X}) - a)^2 + (D(G(\mathbf{z})) - b)^2$$
$$\mathcal{L}_G = \frac{1}{N}\sum_{i=1}^{N}(D(G(\mathbf{z})) - c)^2 \tag{14}$$

which under certain conditions imposed on the relations between a, b and c, can be shown to minimise the Pearson χ^2 divergence [Mao et al., 2017] between the two distributions. In our experiments, we use the suggested values of $a = c = 1$ and $b = 0$. Note that the sigmoid function is not applied to the outputs of either D or G. The other two loss functions we trial focus on minimising the Wassertstein distance between the distributions, the first being the Wasserstein GAN (WGAN) [Arjovsky et al., 2017] with the loss function:

$$\mathcal{L}_D = \frac{1}{N}\sum_{i=1}^{N} -D(\mathbf{X}) + D(G(\mathbf{z}))$$
$$\mathcal{L}_G = -\frac{1}{N}\sum_{i=1}^{N} D(G(\mathbf{z})) \tag{15}$$

Here the expected values of outputs by applying D and G to subsets of data are minimised/maximised in order to increase the expected separation between $D(\mathbf{X})$ and $D(G(\mathbf{z}))$. However to ensure that this loss is continuous and differentiable almost everywhere, the K-Lipschitz condition must be enforced on D and this is achieved by restricting the magnitudes of all weights in D below some value (here we use the recommended value of 0.01). This is a weaker form of enforcing the condition, as clipping the weights does not prevent the gradients of the loss function from diverging (exploding) or vanishing and hence [Gulrajani et al., 2017] proposes another loss function, WGAN-GP which enforces a gradient penalty on D:

$$\mathcal{L}_D = \mathcal{L}_D^{WGAN} + \frac{\lambda}{N}\sum_{i=1}^{N} \nabla_{\mathbf{X}}\left[D(\mathbf{X}) - 1\right]^2$$
$$\mathcal{L}_G = \mathcal{L}_G^{WGAN} \tag{16}$$

which naturally prevents both exploding and vanishing gradients and encourages the norms of the gradients to stay near 1, more strongly enforcing the Lipschitz condition. The choice of lambda must be large enough to have an appropriate level influence on the optimisation process, and here we use the default value of 10 provided by the original authors. We compare these models and we apply a modification to enhance the stability and convergence of LSGANs in section 4.3.

With GANs however, the capacity of network required to have an effective and helpful discriminator, D, to learn from is particularly large relative to the generator, G. Because of this we decided it would be best to train our GAN to only generate one feature. We could therefore choose an arbitrary feature and using a discriminator with unit input and output sizes and a generator with a l-dimensional latent space, we can generate random samples from marginal distributions $p_i(x)$ for $x \in \mathbb{R}$ of this chosen feature, i. This is much less computationally expensive than training a much larger GAN to learn the multivariate distribution of all the features, $p(\mathbf{X})$ for $X \in \mathbb{R}^m$. Whilst the argument can be made that the required capacity increases sub-linearly with the number of features generated, we decided it would be best to proceed with generating a single feature. We also discuss the consequences of this approach in section 4.4.

This constraint on the output size of the network however proposes a practical issue as now the latent space of G should be of dimension 1 if $m \geq l$ is to hold, however [NichG, 2016] suggests that in the case of learning low-dimensional distributions, having a latent space size greater than the size of the space of the data helps prevent the data manifolds from intersecting or tangling in the neural networks hidden layer space. If G can sample from a relatively high-dimensional latent space, then it will be far easier for it to learn a mapping to the data distribution.

The second step and third steps were not achievable in this project. After many experiments, we failed to develop any extensions to the GAN used in the first step, or any models that could post-process generated outputs that would result in "realistic" time-series. The main issue was the lack of metric that could be used to assess how realistic a time series is compared to real outputs. For tasks such as image/video generation or text generation, there are other metrics both qualitative and quantitative to assess realism such as style loss [Gatys et al., 2015] or connectionist temporal classification loss (CTC) [Graves et al., 2006]. No such measures exist for multivariate time-series. We propose possible solutions to these issues in section 4.4, but no justified approach has been found as of yet.

4.2 Model architecture

Here the objective was to learn a time-independent distribution of the data, and hence no recurrent layers were used.

Instead, for both the generator and the discriminator we simply use 4 linear fully connected layers but with varying shapes. For the discriminator, we have an input size of 1 which is expanded to a maximum of 300 units wide in the third layer and then compressed back to an output size of 1. For the generator, we follow a similar pattern with the exceptions that the input size is 20 (having a larger latent space then data space) and the maximum width is 60. It is worth noting that the performance achieved is due to the discriminator being of much higher capacity than the generator.

The LeakyRelu non-linearity was applied to all but the output layers of both networks, with a negative gradient of 0.01. For the standard GAN model only, the sigmoid activation function is applied to the output of the discriminator to smoothly restrict its output to $(0,1)$ is applied to the first of these linear layers. The weights of these networks are initialised according to the Xavier uniform scheme [Glorot and Bengio, 2010], and for the WGAN the weights are clipped at a maximum magnitude of 0.01. For the number of input features specific to this task, a latent space size of 20 was found to a good but not minimal size for producing a good distribution. For training, after optimising hyper-parameters via grid searches, the optimiser used for GAN and LSGAN models was Adam [Kingma and Ba, 2014] with a learning rate of 5×10^{-7} and a weight decay parameter of 0.01. For WGAN and WGAN-GP models, we used the RMSprop optimiser, with the learning rate for the discriminator being 5×10^{-6} and for the generator being 10^{-6}.

4.3 Results

The choice of GAN was not easy to make, with all models demonstrating success to various degrees with learning one dimensional distributions of single features. The least squares loss based GAN displayed superior abilities in fitting to the distribution of the real data (see table 2) whilst being computationally affordable. However we did observe the issue of stalling in learning for GAN and LSGAN models at various intervals during training, as well as abnormal behaviour in the loss over time of the models as seen in figure 16, tested over a number of learning rates and optimisers. We experimented with suggestions from [Salimans et al., 2016] such as label smoothing and label noise yet found no significant improvement. To overcome this issue, we applied a modification to GAN and LSGAN models by adding a $L2$-penalty (weight decay) to the weights of the generator networks:

$$\begin{aligned}\mathcal{L}_D &= \frac{1}{N}\sum_{i=1}^{N}(D(\mathbf{X})-a)^2+(D(G(\mathbf{z}))-b)^2 \\ \mathcal{L}_G &= \frac{1}{N}\sum_{i=1}^{N}(D(G(\mathbf{z}))-c)^2+\gamma\sum_i w_i^2\end{aligned} \tag{17}$$

in the least-squares case, w_i is the set of all weights in G and $\gamma \in \mathbb{R}$ is a parameter that controls the influence this has on the loss function. We find that $\gamma = 0.01$ results in the best convergence for our models. After testing with these, we observe a notable improvement in the convergence and loss function behaviour (see figure 17), but particularly in the reliability of the model, in that we see these improved behaviours almost every time. The reason for applying weight decay was to prevent any generator weights from diverging, whilst also applying a form of gradient penalty since the gradients are weighted by the values of the weights themselves.

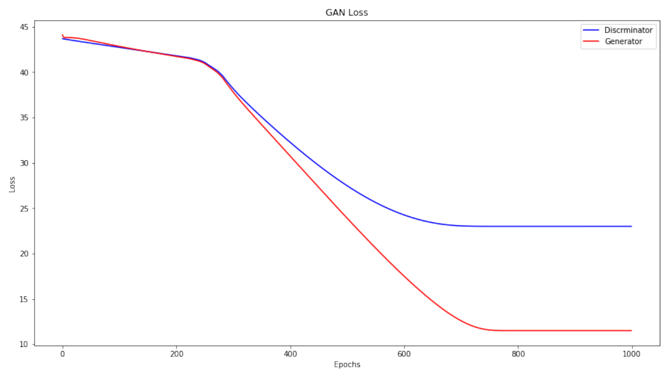

Fig. 16: Loss over time of a LSGAN model, the discriminators loss is in blue and the generators loss is in red. We see here the discriminator continues improving and stalls, whilst the generator loss continues to decrease and stalls shortly after. Such behaviour results in exceptionally poorly fitted generated distributions.

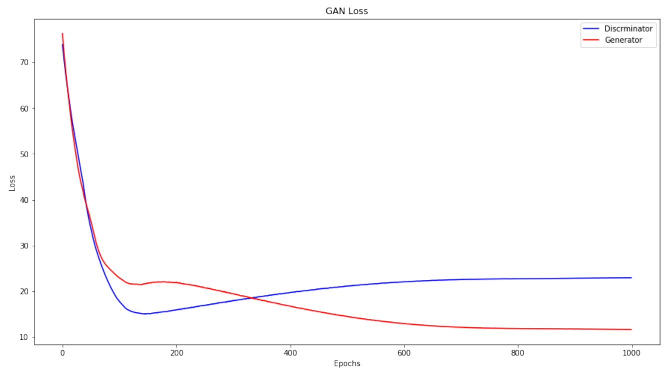

Fig. 17: Loss over time of an ideal LSGAN model with weight decay on the generator. The discriminators loss is in blue and the generators loss is in red. We see here that the discriminator initially improves but gets worse as the generator improves steadily. This behaviour is indicative of a well fitted generator that has not only minimised its own loss but is also capable of reliably fooling the discriminator.

Table 2: Comparison of various GANs with the same architecture (described in section 4.2). We see that the Least squares GAN clearly achieves the best performance in terms of minimising both the Jensen-Shannon divergence (JS-div.) and the Wasserstein distance (W-dist.) between the real and generated distributions.

	GAN	LSGAN	WGAN	WGAN-GP
JS-div.	0.089	**0.0034**	0.014	0.013
W-dist.	197.01	**15.40**	80.92	46.57

After applying these modifications and optimising for various hyper-parameters, we find that the LSGAN models show the best results for us. We see in table 2 that whilst the WGAN and WGAN-GP models both perform well, the least-squares GAN clearly achieves a much better minimum in both metrics used. Figure 18 compares the real and generated distributions and shows how the two are visually identical.

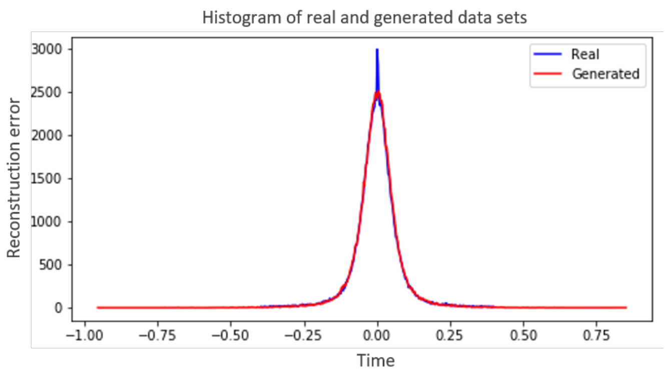

Fig. 18: Histograms of the distributions of data over all time of feature No. 27 (Input pressure). The blue is the histogram created from the real, recorded data and red is there distribution of an equivalent number of samples generated by the generator (multi-layer perceptron) trained using the least-squares loss. The Jensen-Shannon divergence between the two distributions is 0.0034.

4.4 Conclusions

We were able to show that a variety of adversarially trained models with the same architecture were able to learn the probability distribution of the real data available to us, and we were able to sample from a reasonably good approximation to this distribution. However, we were not able to post-process the outputs in order to generate anomalous data but propose possible methods in the next sub-section.

4.5 Further work

Our first objective was to train a generative model to learn the distribution of any given sensor over all of time, but the approach here involved training models to generate and discriminate only one feature per model. To extend this to multivariate distributions for multiple features would require more resources than are currently feasible given the number of experiments required to run and amount of memory required per architecture. If resources were available, then the modelling of joint distributions would only require a sub-linear increase in the capacity of models to learn the distributions due to the non-zero correlations between features. Additionally, we would be able to generate a larger variety of data as real anomalous data could include instances where certain sets of features behave in an anomalous manner whereas others do not, and certain features can only behave in certain ways others change (such as how pressure must change with flow rate etc. for gasses). Therefore the first matter to address given more resources would be to begin simulating joint distributions.

The second objective, to ensure that the random samples drawn from this generative model to form realistic time series was not achieved. Further work here could be to either explore the merging of these two objectives using less common GAN architectures such as Dual discriminator GAN [Nguyen et al., 2017]. With such a model, there would be two discriminators each optimised to distinguish an output from a single generator according to a different criterion. We attempted to have one discriminator minimising the least-sqaures loss as described in section 4.1, and another minimising various forms of explicit ”style” loss or similarity measure between the real and generated time-series instances themselves. We could then take a weighted mean of the two losses:

$$\mathcal{L}_{tot} = \alpha \mathcal{L}_{ls} + \omega \mathcal{L}_{style} \tag{18}$$

where $\mathcal{L}_{ls}$ is the least-squares loss as described in section 4.1 and $\mathcal{L}_{style}$ could be defined using some style or similarity metrics. One such metric is DTW (dynamic time warping) which can be used as a loss function to enable comparisons of similarity between two arbitrary time-series. [Cuturi and Blondel, 2017] provides a efficient implementation of SoftDTW which is a differentiable version of the DTW loss, and hence can be used for the optimisation of parameters with PyTorch.

Other alternatives include multiple-objective GANs [Durugkar et al., 2016] with recurrent layers in one of the discriminators and the generator, or even cycleGANs [Zhu et al., 2017] that could learn some form of implicit ”style” loss between generated and real time series and optimise a recurrent generator.

Another approach could be to have a separate and distinct model that would post-process a N-length set of random samples from the generative model into a "realistic" time-series. First of all, we would need some metric to determine what constitutes a realistic time-series in this context, and so options that could be explored further include DTW (dynamic time warping) as a loss function to enable comparisons of similarity between two arbitrary time-series. [Cuturi and Blondel, 2017] provides a efficient implementation of SoftDTW which is a differentiable version of the DTW loss, and hence can be used for the optimisation of parameters with PyTorch.

Using this or other metrics, it may be possible to train models such a de-noising autoencoders [Vincent et al., 2008] to post-process random samples from a given distribution to result in synthetic data with statistics that can approximate the statistics of real data. The theoretical justification for this particular model is lacking however. These models are known to be able to remove any form of noise ε from an input signals of the form $x+\varepsilon$ or even εx given enough capacity, data and optimisation, but if the input is a random sample then it's unclear how an input with no underlying signal will be processed by such a de-noising autoencoder.

5 References

References

[Ahmed et al., 2016] Ahmed, M., Mahmood, A. N., and Islam, M. R. (2016). A survey of anomaly detection techniques in financial domain. *Future Generation Computer Systems*, 55:278–288.

[Arjovsky et al., 2017] Arjovsky, M., Chintala, S., and Bottou, L. (2017). Wasserstein generative adversarial networks. In *International conference on machine learning*, pages 214–223.

[Bates et al., 2017] Bates, A. B., Rahilly, P., and Macnab, S. (2017). System and methods for automated plant asset failure detection. US Patent 9,535,808.

[Bengio et al., 2009] Bengio, Y. et al. (2009). Learning deep architectures for ai. *Foundations and trends® in Machine Learning*, 2(1):1–127.

[Caruana and Niculescu-Mizil, 2006] Caruana, R. and Niculescu-Mizil, A. (2006). An empirical comparison of supervised learning algorithms. In *Proceedings of the 23rd international conference on Machine learning*, pages 161–168. ACM.

[Cortes and Vapnik, 1995] Cortes, C. and Vapnik, V. (1995). Support-vector networks. *Machine Learning*, 20(3):273–297.

[Cuturi and Blondel, 2017] Cuturi, M. and Blondel, M. (2017). Soft-dtw: a differentiable loss function for time-series. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 894–903. JMLR. org.

[Durugkar et al., 2016] Durugkar, I., Gemp, I., and Mahadevan, S. (2016). Generative multi-adversarial networks. *arXiv preprint arXiv:1611.01673*.

[Esteban et al., 2017] Esteban, C., Hyland, S. L., and Rätsch, G. (2017). Real-valued (medical) time series generation with recurrent conditional gans. *arXiv preprint arXiv:1706.02633*.

[Gatys et al., 2015] Gatys, L. A., Ecker, A. S., and Bethge, M. (2015). A neural algorithm of artistic style. *arXiv preprint arXiv:1508.06576*.

[Glorot and Bengio, 2010] Glorot, X. and Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256.

[Goodfellow et al., 2014] Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2014). Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680.

[Graves, 2013] Graves, A. (2013). Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850*.

[Graves et al., 2006] Graves, A., Fernández, S., Gomez, F., and Schmidhuber, J. (2006). Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks. In *Proceedings of the 23rd international conference on Machine learning*, pages 369–376. ACM.

[Gugulothu et al., 2017] Gugulothu, N., TV, V., Malhotra, P., Vig, L., Agarwal, P., and Shroff, G. (2017). Predicting remaining useful life using time series embeddings based on recurrent neural networks. *arXiv preprint arXiv:1709.01073*.

[Gulrajani et al., 2017] Gulrajani, I., Ahmed, F., Arjovsky, M., Dumoulin, V., and Courville, A. C. (2017). Improved training of wasserstein gans. In *Advances in neural information processing systems*, pages 5767–5777.

[Haixiang et al., 2017] Haixiang, G., Yijing, L., Shang, J., Mingyun, G., Yuanyue, H., and Bing, G. (2017). Learning from class-imbalanced data: Review of methods and applications. *Expert Systems with Applications*, 73:220–239.

[Hinton and Zemel, 1994] Hinton, G. E. and Zemel, R. S. (1994). Autoencoders, minimum description length and helmholtz free energy. In *Advances in neural information processing systems*, pages 3–10.

[Hochreiter and Schmidhuber, 1997] Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, 9(8):1735–1780.

[Kingma and Ba, 2014] Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

[Lin and Lin, 2003] Lin, H.-T. and Lin, C.-J. (2003). A study on sigmoid kernels for svm and the training of non-psd kernels by smo-type methods. *submitted to Neural Computation*, 3:1–32.

[Liu et al., 2008] Liu, F. T., Ting, K. M., and Zhou, Z.-H. (2008). Isolation forest. In *2008 Eighth IEEE International Conference on Data Mining*, pages 413–422. IEEE.

[Mao et al., 2017] Mao, X., Li, Q., Xie, H., Lau, R. Y., Wang, Z., and Paul Smolley, S. (2017). Least squares generative adversarial networks. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2794–2802.

[Mogren, 2016] Mogren, O. (2016). C-rnn-gan: Continuous recurrent neural networks with adversarial training. *arXiv preprint arXiv:1611.09904*.

[Münz et al., 2007] Münz, G., Li, S., and Carle, G. (2007). Traffic anomaly detection using k-means clustering. In *GI/ITG Workshop MMBnet*, pages 13–14.

[Nguyen et al., 2017] Nguyen, T., Le, T., Vu, H., and Phung, D. (2017). Dual discriminator generative adversarial nets. In *Advances in Neural Information Processing Systems*, pages 2670–2680.

[NichG, 2016] NichG (2016). Building a 1D Generative Adversarial Network in TensorFlow. [Online; accessed 26-August-2019].

[Park et al., 2018] Park, D., Hoshi, Y., and Kemp, C. C. (2018). A multimodal anomaly detector for robot-assisted feeding using an lstm-based variational autoencoder. *IEEE Robotics and Automation Letters*, 3(3):1544–1551.

[Rüping, 2001] Rüping, S. (2001). Svm kernels for time series analysis. Technical report, Technical Report, SFB 475: Komplexitätsreduktion in Multivariaten .

[Salimans et al., 2016] Salimans, T., Goodfellow, I., Zaremba, W., Cheung, V., Radford, A., and Chen, X. (2016). Improved techniques for training gans. In *Advances in neural information processing systems*, pages 2234–2242.

[Schlegl et al., 2017] Schlegl, T., Seeböck, P., Waldstein, S. M., Schmidt-Erfurth, U., and Langs, G. (2017). Unsupervised anomaly detection with generative adversarial networks to guide marker discovery. In Niethammer, M., Styner, M., Aylward, S., Zhu, H., Oguz, I., Yap, P.-T., and Shen, D., editors, *Information Processing in Medical Imaging*, pages 146–157, Cham. Springer International Publishing.

[Tieleman and Hinton, 2012] Tieleman, T. and Hinton, G. (2012). Rmsprop: Divide the gradient by a running average of its recent magnitude. coursera: Neural networks for machine learning. *Tech. Rep., Technical report*, page 31.

[Vincent et al., 2008] Vincent, P., Larochelle, H., Bengio, Y., and Manzagol, P.-A. (2008). Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th international conference on Machine learning*, pages 1096–1103. ACM.

[White, 2016] White, T. (2016). Sampling generative networks. *arXiv preprint arXiv:1609.04468*.

[Xu et al., 2015] Xu, B., Wang, N., Chen, T., and Li, M. (2015). Empirical evaluation of rectified activations in convolutional network. *arXiv preprint arXiv:1505.00853*.

[Zhou et al., 2016] Zhou, S., Shen, W., Zeng, D., Fang, M., Wei, Y., and Zhang, Z. (2016). Spatial–temporal convolutional neural networks for anomaly detection and localization in crowded scenes. *Signal Processing: Image Communication*, 47:358–368.

[Zhu et al., 2017] Zhu, J., Park, T., Isola, P., and Efros, A. A. (2017). Unpaired image-to-image translation using cycle-consistent adversarial networks. *CoRR*, abs/1703.10593.